International Journal of Computer Science Engineering
and Information Technology Research (IJCSEITR)
ISSN (P): 2249-6831; ISSN (E): 2249-7943
Vol. 9, Issue 1, Jun 2019, 1-10

THE SCHEME PROGRAMMING LANGUAGE

SMITA DESAI[1]& SHREYA DESAI[2]

[1]*Bharatesh College of Computer Applications, Belagavi, Karnataka, India*

[2]*Chalmers, Gothenburg, Sweden*

ABSTRACT

This paper is a review of the Scheme programming language. Origin of Scheme, various features, syntax, example code and applications of Scheme language are discussed in this paper. The interpreters that are used and other dialects of LISP are also discussed in this paper. The purpose of this paper is to create an awareness of the Scheme programming language to the programmers.

KEYWORDS: Scheme, Ai, Lisp, Functional Programming, Drracket & Hygienic Macrosintroduction

Received: Dec 4, 2018; **Accepted:** Jan 14, 2019; **Published:** Jan 31, 2019; **Paper Id.:** IJCSEITRJUN20191

Original Article

INTRODUCTION

The scheme is a highly expressive high level language. It is a dialect of LISP language and supports multi-paradigm functional and procedural programming. It was designed by Guy L. Steele and Gerald Jay Sussman in 1970. Scheme was introduced as Sussman and Steele Lambda Papers. It was the first Lisp dialect to choose a static variable scope over dynamic variable scope and was one of the first programming languages which supported first-class continuations. The standardization of the scheme began in 2003 and in 2006 it was standardized as R6R5 standard. R6R5 standard allows the split between core language and libraries.

FEATURES OF SCHEME

Scheme as a Minimalist Language

The scheme is known as minimalist language. It has a reach data set making it extremely versatile. Scheme programs are highly portable on various platforms. Dr Racket is the IDLE used for Scheme programming. It has minimal syntax like all other Lisp dialects. Its fully nested and parenthesized notation rules out the operator precedence. The new syntactic constructs can be added to the language by using Scheme's powerful macro system. The lexical scoping avoids the risk of common programming error that can occur in macro systems of other programming languages. User can create non-local constructs using *call-with-current-continuation* that must be built into other languages like backtracking, iterators etc.

The scheme was one of the first programming languages that used first class procedures as in lambda calculus. Thus, incorporating the block structure and static scope rules in a

- **DrRacket IDE**

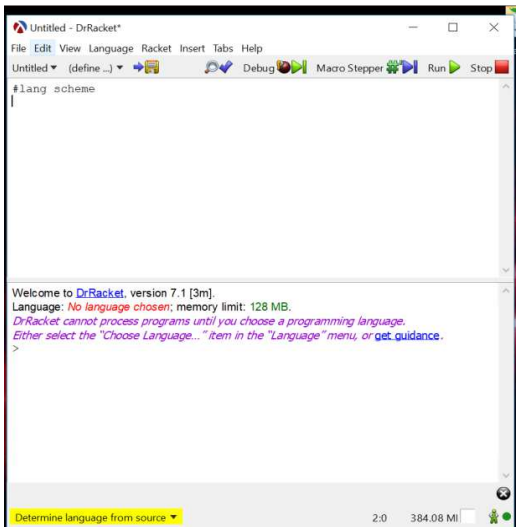


Figure 1: Dr Racket IDE

- **Comments**

The short comments on a line in Scheme start with a semicolon (;)

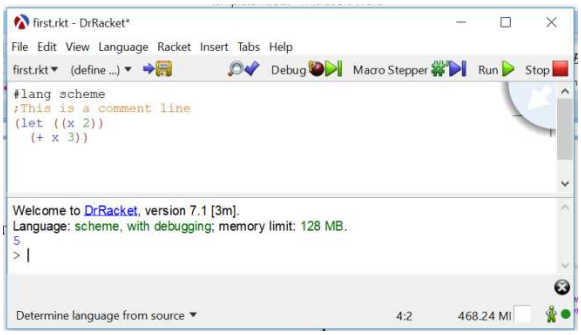


Figure 2: Code Example, with Comment Line

Two semicolons (;;) are used in a comment within a function on its own line and three semicolons (;;;) are used global or introductory comments, outside a function definition.

A #; in Dr Racket will treat the entire code followed by it as a comment, as in Figure 2

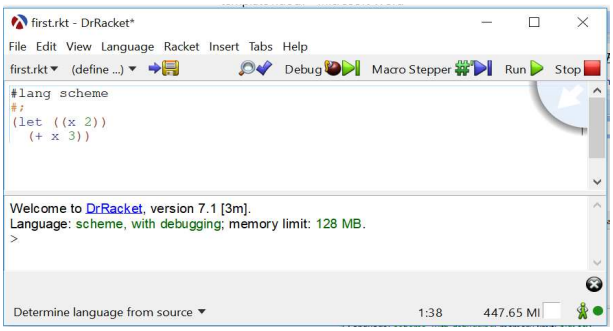


Figure 3:Comment with #; in Scheme

For extended comments in Scheme we use #| and |# as in Figure 3.

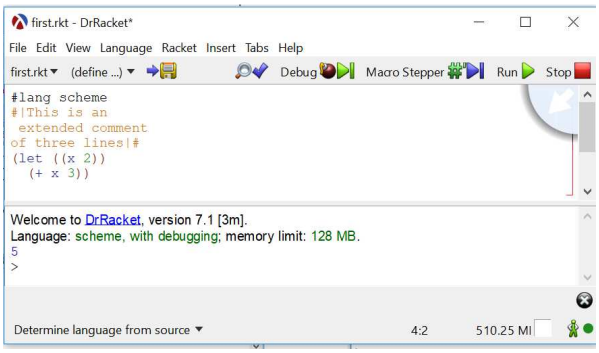


Figure 4: Multi Line Comments in Scheme

- **Variables**

The scheme has both local as well as global variables. A variable is a name bound to some data objects using a pointer. The scheme does not have any type declarations for variables. Variables are bound using the special form to define.

(define symbol(variable) expression)

define binds the symbol to the result of evaluating expression. The following line declares a variable called 'a' and makes refer to 10.

(define a 10)

a =>10

(define a 20) ;this rebinds a to 20

(+ a 2) =>22

(define l '(a b c))

(define p '(a b c))

Here both l and p are pointers, both pointing at the same list.

- **Lexically Scoped Variables with Let and Let***

Scheme uses special form let to declare and bind local variables.

For example:

(let ((variable1 value1)

(variable2 value2)

…

(variableN valueN))

For example,

;; the efficient code to reverse the list and double it

(define r1 x)

(let ((r (reverse x)))

(append r r)))

Note: The problem with bindings created with let is that expressions cannot refer to bindings that have been made previously. For example, in the following code x is not known outside the body.

(let ((x 5)

(y (+ x 1)))

(+ x y))

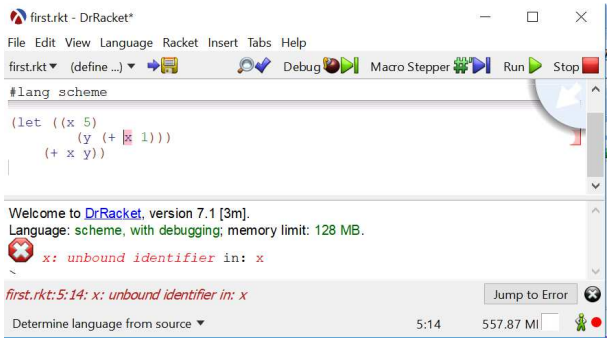


Figure 5: Scheme Program Using Let

- **Functions**

a. Anonymous Function: Lambda

Scheme uses special form *lambda* to create anonymous functions. See the following example:

(lambda (param1 param2.. paramk);list of formals

expr) ;body

lambda expression evaluates to an anonymous function. When we execute lambda function, it takes k arguments and returns the results of devaluating expr. The parameters are lexically scoped and can only be used in expr.

For example:

(lambda (x1 x2)

(* (- x1 x2) (- (x1 x2)))

The result of lambda expression can be directly applied by providing arguments, as in following example,

((lambda (x1 x2)

(* (- x1 x2) (- x1 x2)))

(2-5) ; evaluates to 49

b. Named Functions in Scheme

1)You can bind the result of a lambda to a variable using define(first class functions).

For example:

(define sq-plus

(lambda (x1 x2)

(* (+ x1 x2) (+ x1 x2))))

Scheme provides a special shortcut version of *define* that does not use *lambda* explicitly.

(define (function0name param1 param2.. paramk)

expr)

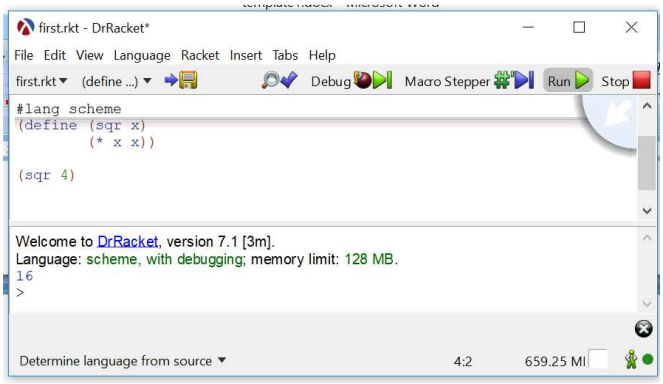


Figure 6: Has Some Examples of Define

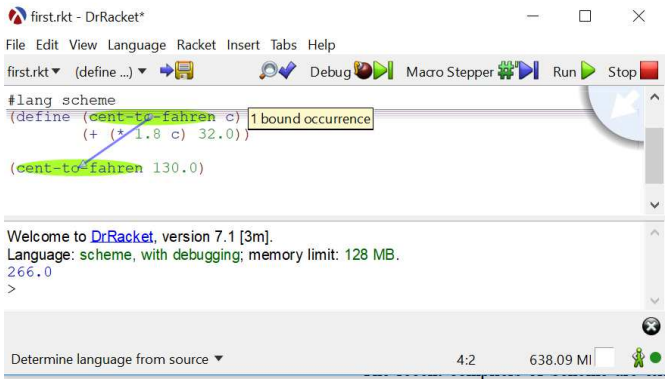


Figure 7: Using Define without Lambda Function

The scope of x in sqr function is only within the function sqr. Functions can have zero(0) arguments as follows,

(define (fn) 5)

(fn) => 5

- **Lists**

The most important built in data type in Scheme. Lists are unbounded, heterogeneous collections of data.

Examples of list,

(x)

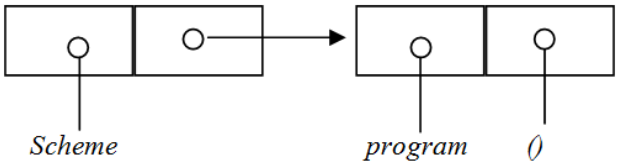
(scheme program)

(2 4 6 8 10)

(2 4 x y 3.4 “language”)

()

List Representation



Note :

- (x) is not same as x
- () is the empty list
- List of lists: ((a b) (c d))

Data Types

Following is the list of primitive data types in Scheme:

- Numbers
  - Integers(e.g.1, 7, -5)
  - Real numbers(e.g. 3.2,8.6,2.32E10)
  - Rational numbers(e.g. 4/3,5/2)
- Symbols (e.g. x,a1,s!,abc)
- Boolean : Scheme has special Boolean symbols #f and #t to represent false and true respectively
- Srtings (e.g. “hello”,”world”)
- Characters(e.g. #\c)

Equality

The three primitives in Scheme for equality and identity testing are:

- **eq? –** used for pointer comparison. It returns #t iff its arguments refer to the same objects in memory. Two variables that refer to the same object are **eq.**
- **eqv? –** is like **eq?**, but differs only for the number and character data types from **eq?**
- equal? – it returns #t iff arguments are eqv. It also returns #t if its arguments are lists with corresponding elements

equal.

- **eq** is also called as identity comparison and **equal** is called as equality comparison.

Control Structures

if special form:

if does not automatically evaluate all of its arguments.

(if condition true_expression false_expression)

If the condition is true, then true_expression is returned alse false_expression is returned.

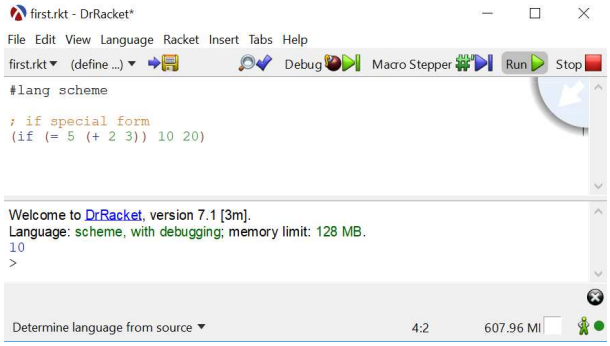


Figure 8: If Special form in Scheme

The condition in Figure 7 evaluates to true, hence true expression i.e. 10 is returned.

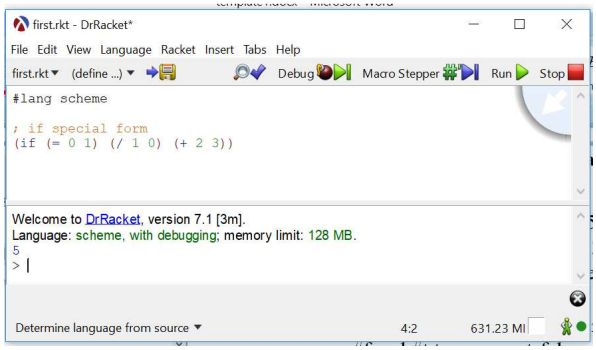


Figure 9: If Condition Evaluating False

The condition in Figure 8 evaluates to false, hence false expression i.e. (+ 2 3) => 5 is returned

cond

the general form of cond special form:

(cond (test1 expr1)

(test2 expr2)

…

(else exprn))

As soon as the test evaluates to true, the corresponding expr is evaluated and its value is returned. If none of the

tests evaluates to true then the else part is evaluated(exprn).

- **Loops**
  - **while loop** – is very simple and roughly analogous to while loop in C.
  - **do loop** – this is a powerful looping construct in Scheme. It is roughly analogous to for loop in C.
  - **map loop** – map loop iterate over a list(array).

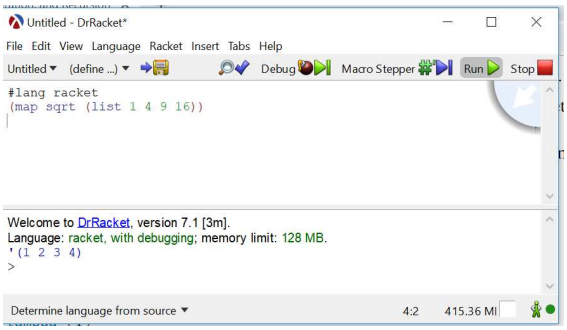


Figure 10: Map Loop Example

- **Input/Output**

  Read-line procedure reads one line that user types and returns it as a string. display is used to display the output.

  Refer Figure 10 for input/output in Scheme

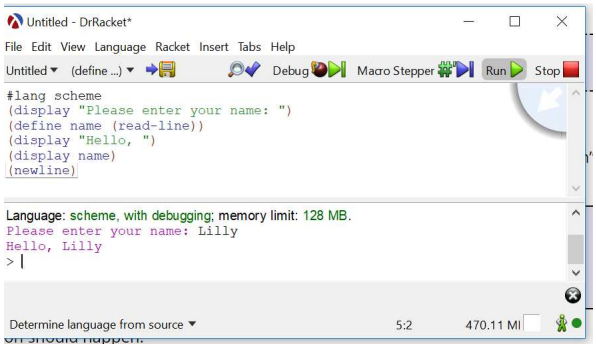


Figure 11: Illustrating the Input/Output in Scheme

IMPLEMENTATION OF SCHEME

The scheme is typically used to analyse and optimize application based compilers, to write text editing programs, to write drivers for graphic cards, to develop operating systems and much more. It can also be used to develop commercial applications that are numerically based, to calculate large finance based applications, to analyse financial resources and much more. The scheme is also being used for the development of Virtual Reality.

PERFORMANCE AND EFFICIENCY

The recent compilers of the Scheme are extremely efficient and fast. These programs run at par with programs written in low-level languages (almost near to assembly code). The scheme code implementations are extremely fast for example Chez scheme code. Though it is fast, optimizing the Scheme code is not easy. Writing a compiler code in c is

more easy than writing a compiler code in Scheme. The scheme is complicated because of higher order functions, data flow and type checking etc.

REFERENCES

1. *Bohdan B Khomtakchouk Edmund Weitz Peter D Karp Claes Wahlestedt,"How the strength of Lisp-family languages facilitate building complex and flexible bioinformatics applications.*
2. *Sadek, H. A., Khami, M. J., & Obaid, T. A. (2013). Computer Simulation of Blood Flow in Large Arteries by a Finite Element Method. International Journal of Computer Science and Engineering (IJCSE), 2(4), 171-184.*
3. *R. Kent Dybvig, "The Scheme Programming Language", Third Edition.*
4. *R. Kent Dybvig, "The Scheme Programming Languages", Fourth Edition.*
5. *https://cacm.acm.org/magazines/2018/3/225475-a-programmable-programming-language*
6. *https://academic.oup.com/bib/article/19/3/537/2769437*

Author Details

REYYA, S., Prameela, M., YADAV, G. V., RANI, K. S., & Bhargavi, A. V. An efficient extension of conditional functional dependencies in distributed databases. *database*, *5*(6).

Mrs. Smita Desai

Completed MCA, has rich working experience for past 20 yearsin the field of education. Has served as Training officer, Principal, Corporate trainer at various bodies. To name few are Apple Industries Ltd. Mumbai, TechTree IT Systems, Bangalore, Tarun Bharat Daily, Belgavi, MTNL, Mumbai. Currently working as Vice-Principal at Bharatesh College, of Computer applications Belagavi, Karnataka, India.

Miss. Shreya Desai

Completed Bachelor of Engineering in Electronics and Communications. Presently pursuing M.S. in Computer Systems and Networking, Chalmers University, Gothenburg, Sweden.